Security Analysis of an Open Car Immobilizer Protocol Stack

Stefan Tillich and Marcin Wójcik

University of Bristol, Computer Science Department, Merchant Venturers Building, Woodland Road, BS8 1UB, Bristol, UK
{tillich,wojcik}@cs.bris.ac.uk

Abstract. Openness is a key criterion of security algorithms and protocols which enable them to be subjected to scrutiny by independent security experts. The alternative "methodology" of secret proprietary algorithms and protocols has often ended in practical breaks, e.g. of the MIFARE Oyster cards for public transport or the KeeLoq remote control systems. Open evaluation is common for general applications of security, e.g. the NIST competitions for selection of the Advanced Encryption Standard (AES) and the Secure Hash Algorithm 3 (SHA-3). Nowadays an increasing number of embedded security applications apply the principle of open evaluation as well. A recent example is the specification of an open security protocol stack for car immobilizer applications by Atmel, which has been presented at ESCAR 2010. This stack is primarily intended to be used in conjunction with automotive transponder chips of this manufacturer, but could in principle be deployed on any suitable type of transponder chip. In this paper we analyze the security of this protocol stack. We were able to uncover a number of potential security vulnerabilities, for which we suggest fixes.
Keywords: Security, car immobilizer, algorithms, protocols, openness, analysis.

1 Introduction

Securing systems through secrecy of the involved algorithms and protocols is not always successful. Often, once the details of the algorithm have been disclosed through various channels, practical attacks quickly become possible, e.g. on the MIFARE Oyster card for the London transport system [6] or the KeeLoq algorithm used in remote control systems [7]. In contrast, subjection of cryptographic methods to public scrutiny is a widely accepted method of preventing such breaks during deployment. Prominent examples of this strategy are the Advanced Encryption Standard (AES) competition [11] and the Secure Hash Algorithm-3 (SHA-3) competition [12]. In this paper we analyze the security of a car immobilizer protocol stack which is facilitated by its openness.

A car immobilizer is a system that requires the presence of a security token (often in the form of a key fob) to allow a car to run. If this token is not present, the car's Engine Control Unit (ECU) interrupts key components like the ignition,

the starter motor circuit, or the fuel pump. The communication between car and key fob is typically done via RFID, where the car is fitted with an RFID reader and the key fob contains an RFID tag. While earlier models used a static code in the key fob, modern immobilizers utilize either rolling codes or cryptography to prevent duplication of the key fob. Communication between car and key fob involves the use of a protocol stack which defines frame sizes, data formats, error detection, data transformations, etc.

An open security protocol stack for car immobilizer applications has been presented in [8]. It is mainly intended for use with specific automotive transponder chips. According to [8], the stack consists of a physical layer, a logical layer, a protocol layer, and the AES crypto layer. The physical layer deals with modulation types, data encoding, and bit timing. The logical layer defines the functional behavior of the reader and the transponder and includes communication link controls, controls configuration, setup of functional dependencies and error resolution. The protocol layer allocates data frames and buffers for reading and writing. It implements the user command interface, authentication, and key learning (*i.e.* changing cryptographic keys before and after deployment). The AES crypto layer controls the data authentication results[1]. Both physical and AES crypto layer are already industry standards. The logical and protocol layer, which are usually proprietary, are made open. This means the specification of these layers is available for inspection and modification.

The protocol stack implements a number of commands to be issued by the reader to the key fob. In most cases, the car featuring the immobilizer functionality acts as reader but the reader can also be a programming device used by the car manufacturer or distributor. The communication between reader and key fob uses the LF band at 125 kHz. In this band, the normal read range is usually very limited (commonly a few centimetres), but there are readers available which can extend it to up to one metre [3, 5] and thus allowing for attacks in close proximity of the key fob.

The command set out in the protocol stack's specification [1] encompasses eleven commands. They include reading of the key fob's unique ID (UID) and error status, initiation of authentication, setting of the used secret keys, initiation and leaving of the so-called enhanced mode (for RF communication powered by the battery), a request to repeat the last response, reading and writing of user memory as well as setting memory access protection to certain memory sections. Authentication can be configured to be unilateral (only key fob authenticates itself to the reader) or bilateral (both key fob and reader authenticate themselves to each other). If bilateral authentication is configured, some commands like reading and writing user memory can only be executed when there has been a previous successful authentication.

[1] The description of this protocol layer in [8] probably refers to the use of the AES block cipher in the execution of various commands by reader and key fob. As such it is debatable whether it constitutes a separate layer or should be considered as part of the protocol layer.

Authentication follows the challenge-response pattern [10]. The party who wants to authenticate sends out a challenge (usually a random number) and the other participant transforms the challenge cryptographically using a secret or private key and returns the response. The first party then checks this result using its knowledge of the same secret key or the according public key. The point of the challenge is to prevent replay attacks, where messages recorded from a genuine protocol run are replayed by an attacker at a later time to achieve authentication. Therefore, the challenge must be non-repeating or only repeat with negligible probability.

The investigated protocol stack has the caveat that the key fob is not expected to be able to generate challenges. This is no problem for unilateral authentication, where the challenge is generated by the reader alone, but poses difficulties for bilateral authentication. Bilateral authentication works by reusing the challenge from the reader for the challenge of the key fob. The cryptographic transformation involved in the authentication is AES encryption with one of two shared keys. Figure 1 shows the essential steps of bilateral authentication as given in [1]. Note that the complete authentication also includes the car reading the UID from the key fob in order to allow early termination of the protocol when a wrong key fob is accidentally in read range. We have omitted this part of the authentication protocol as it is not of interest from a security point of view.

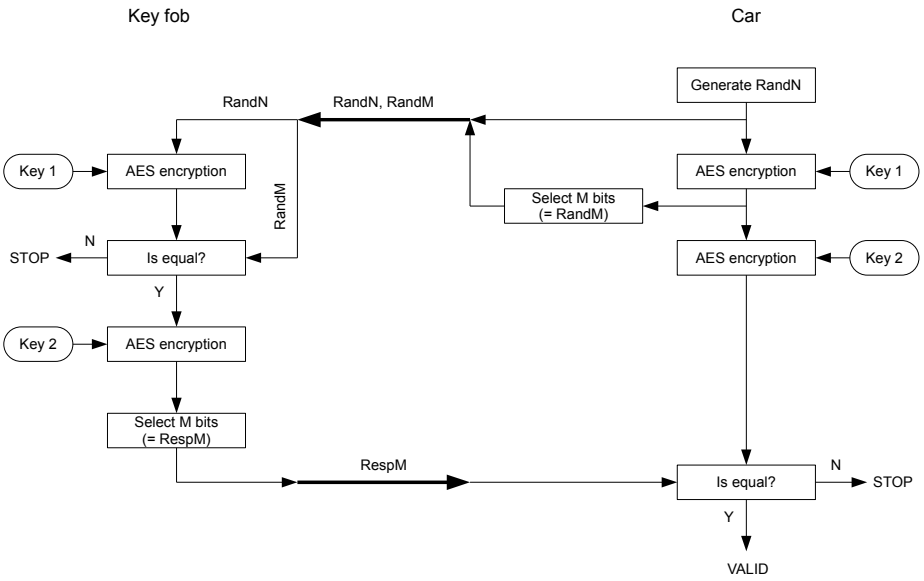

Fig. 1: Bilateral authentication between key fob and car.

Car and key fob share two AES keys (Key 1 and Key 2). The car generates the N-bit challenge RandN, encrypts it with Key 1 and selects M bits of the

resulting ciphertext as RandM. N and M can be configured to be less than the AES block size of 128 bits in order to reduce communication overhead. RandN and RandM are sent to the key fob, which validates that RandM originated from RandN via encryption with Key 1. If this is successful, the car is authenticated to the key fob. The key fob uses the output of the first AES encryption as input for a second AES encryption with Key 2. As this value is not fully known to an eavesdropper (M being usually smaller than 128), it is also denoted as hidden challenge. M bits of the second encryption result are selected as RespM, which is sent to the car. The car then verifies that RespM resulted from encryption with Key 2. On success, the key fob is authenticated to the car and bilateral authentication is finished.

2 Tracking

The protocol stack includes the "ReadUID" command to retrieve the 32-bit UID from the key fob. There is no security mechanism in place which would require authentication by the reader. Therefore, any reader can request the UID and the key fob can be potentially tracked via a number of readers installed at various places.

Tracking could be prevented if the UID is not returned in cleartext, but dependent on a shared secret and a nonce. A simple example is to use the existing AES encryption E_K with one of the pre-shared keys K in a tweakable block cipher construction $\tilde{E}_K$ [9].

$$\tilde{E}_K(\text{nonce}, \text{UID}) = E_K(\text{nonce} \oplus E_K(\text{UID})) \tag{1}$$

The result of $\tilde{E}_K$ will vary with the nonce and the UID will be protected even when the nonce is revealed. Thus, even though the key fob can be still queried by any reader, the result cannot be used any more to track it.

There are two options for the values returned by the key fob depending on the actual functional requirements. If the complete result of $\tilde{E}_K$ is returned alongside with the nonce, the reader can decrypt it and arrive at the original UID. Thus, the full functionality of the original "ReadUID" command is retained. This comes at the price of a relatively high communication overhead as the key fob needs to send the 128-bit ciphertext $\tilde{E}_K$ and the nonce. The computational overhead would essentially be the generation of the nonce and two AES encryptions on the key fob side and two AES decryptions on the reader side.

Alternatively, the reader could still check for a specific UID if only a part of the result of $\tilde{E}_K$ were returned with the nonce. This could be useful if the reader requires the "ReadUID" command exclusively to check for a specific UID. We denote this new command as "CheckUID" and its functionality is shown in Figure 2. It's advantage is a shorter response and a better response time of the key fob compared to the enhanced "ReadUID" command.

By varying the size of the nonce and the portion of $\tilde{E}_K$ to be checked (M-bit RespM), the security and communication overhead can be balanced. For example, using a 32-bit portion of $\tilde{E}_K$ for checking, a similar resilience against

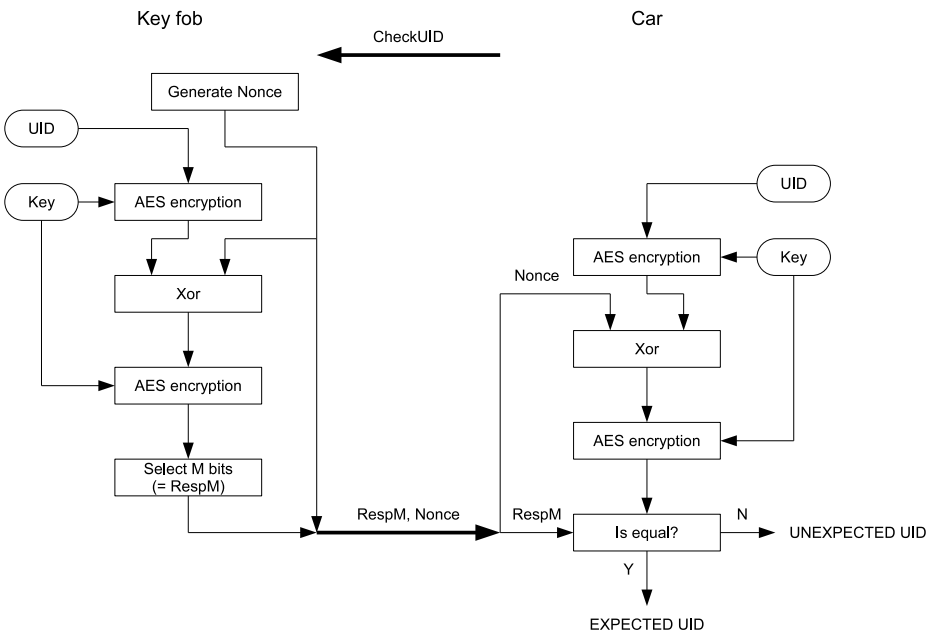

Fig. 2: Enhanced "CheckUID" command with resistance against tracking.

accidentally matching UIDs would be introduced as in the original protocol stack with 32-bit UIDs. The communication overhead would consist of the extra bits of the nonce and the computational overhead would be the generation of the nonce for the key fob and two extra AES encryptions for key fob and reader each. The encrypted UID ($E_K(\text{UID})$) could also be pre-computed and stored which would reduce the computational overhead by one AES encryption for each side.

In both cases, the key fob must be able to generate nonces. This might require a key fob with slightly higher capabilities as set out in the protocol stack specification. Generation of nonces is also required by the countermeasure to the attack described in Section 5.

3 Denial-of-Service Attacks

The protocol stack includes commands for writing new cryptographic keys to the key fob, which replaces the old keys used for authentication. There are two different modes for doing this: In open mode, a "Learn Secret Key1" or "Learn Secret Key2" can be issued by any reader in order to set new keys. In secure mode, an encrypted key is sent by the reader device, decrypted by the key fob and the result is set as new key as shown in Figure 3. The key used for encrypting the new key is the so-called Default Secret Key which is factory set.

Overwriting keys in open mode is trivial, as the malicious reader only has to send the according command to set the keys to those of her choice. However,

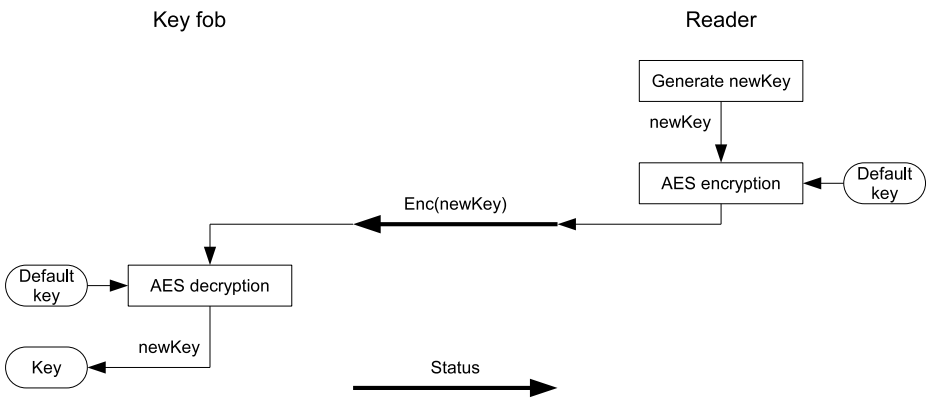

Fig. 3: LearnSecretKey command in secure mode.

even in secure mode it is possible to overwrite keys though the value of the new keys stays hidden to the attacker. This is possible because the secure key learn command only uses the encrypted key but no integrity check for it. Therefore, an attacker can send a random value as encrypted key and the key fob will set the decrypted value as new key.

Thus, in both open and secure mode, keys can be overwritten without the need of knowing a shared secret. Once this has been done, the key fob will no longer work with the car. If the key fob is queried in intervals while the car is in motion, it might even be possible to force the immobilizer to stop the car by overwriting the keys.

The open mode is vulnerable against this attack per design. To defend against the attack in secure mode, a message authentication code (MAC) should be included with the encrypted key and the key should only be overwritten when the MAC is verified successfully. This entails communication overhead for transmission of the MAC from the reader to the key fob and computational overhead of MAC generation in the reader and MAC verification in the key fob.

4 Relay Attack with Genuine Key Fob

Another type of attack tricks the car into thinking that the key fob is in its immediate vicinity when it is actually located further away. Such relay attacks have been known as early as 1976 [2] and have been practically demonstrated, e.g. in [4] for the EMV chip and PIN setting. In the current setting, this attack relays messages between the genuine key fob and the car through a transparent reader (close to the genuine key fob) connected to a transparent key fob (close to the car) as shown in Figure 4. Such an attack would require two cooperating attackers, one bringing the transparent reader close to the genuine key fob and the other gaining entry to the car and bringing the transparent key fob close to the car's reader.

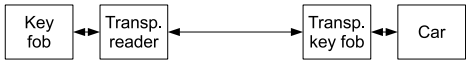

Fig. 4: Relay attack with transparent reader and key fob.

A potential countermeasure to this relay attack is to measure the communication delay between the reader's challenge and the key fob's response in order to detect the actual distance between the communicating endpoints. Alternatively, a dedicated protocol, like the distance bounding protocol used in [4] could be employed. However, the protocol stack includes a mechanism to defeat such countermeasures. If the transparent key fob fakes an uplink CRC error, this forces the car to send a "Repeat Last Response" command. The attacker can use the extra time for the repeated response to get the actual response from the genuine key fob.

This remote attack could be defended against with the measurement of the communication delay of the key fob by the car and by abandoning the mechanism of requesting a repeat of the the key fob's response in answer to a CRC error. Instead the whole sequence of commands and responses should be repeated when a CRC error is encountered. This gives the attacker no time to hide the extra communication delay introduced by the transparent reader and key fob. Measurement of the communication delay might require extra components (e.g. a high-precision oscillator) at the car's side.

5 Replay Attack on Authentication

A unique property of the bilateral authentication protocol in the immobilizer stack is that the key fob is not required to generate nonces. Instead, the encrypted nonce from the reader is "reused" as the challenge from the key fob. While this makes the structure of the key fob simpler, it also means the commands from the reader can be recorded and replayed at a later time to achieve authentication. Thus an attacker can pretend to be an authenticated reader, which gives her access to advanced commands like "Read User Memory" and "Write User Memory".

A defense against this attack is to have the key fob generate the challenges for the reader. Without a challenge from the key fob, the replay of the reader command will lead to a successful authentication of the reader.

6 Spoofing Attack on Memory Access Protection

The protocol stack allows the reader to lock the EEPROM sections AP1 to AP3 via a "Write Memory Access Protection" command. This command is accepted by the key fob without prior authentication. Depending on the actual use of these EEPROM sections, an attacker could impair the functionality of the key fob by locking them with a spoofed command.

By requiring prior authentication for the "Write Memory Access Protection" command this attack can be prevented.

7 Conclusions

In this paper we have identified a number of potential security vulnerabilities in an open car immobilizer stack. The vulnerabilities include tracking of key fobs, denial-of-service attacks to render key fobs useless, achieving key fob authentication despite absence of the key fob (relay attack), achieving reader authentication via a replay attack, and a spoof attack to lock out EEPROM sections of the key fob. For each of the identified vulnerabilities we propose countermeasures. This proves the great value of the openness of the protocol stack to public review. Some of our proposed countermeasures can be implemented rather easily while others require enhanced functionalities from the reader and/or the key fob.

Acknowledgements. The research described in this paper has been supported by EPSRC grant EP/H001689/1. The information in this document reflects only the author's views, is provided as is, and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. Atmel. Open Source Immobilizer Protocol Stack. Available online at `http://www.atmel.com/dyn/products/tools_card.asp?tool_id=17197` (registration required), 2010.
2. J. H. Conway. *On Numbers and Games*. Academic Press, 1976.
3. Daily RFID Co., limited. LF RFID Reader-03. `http://www.rfid-in-china.com/2008-09-06/products_detail_2140.html`.
4. S. Drimer and S. J. Murdoch. Keep Your Enemies Close: Distance Bounding Against Smartcard Relay Attacks. In *Proceedings of the 16th USENIX Security Symposium*, pages 87–102, 2007.
5. GAO RFID Inc. 125 kHz Long Range Reader. `http://www.gaorfid.com/index.php?main_page=product_info&products_id=363`.
6. F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling MIFARE Classic. In S. Jajodia and J. Lopez, editors, *13th European Symposium on Research in Computer Security (ESORICS 2008), Malaga, Spain, 6-8 October, 2008, Proceedings (to appear)*, Lecture Notes in Computer Science. Springer Verlag, 2008.
7. S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on KeeLoq. In N. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.
8. P. Lepek. Configurable, Secure, Open Immobilizer Implementation. In *Proceedings of the 8th Embedded Security in Cars (ESCAR) Conference.*

9. M. Liskov, R. L. Rivest, and D. Wagner. Tweakable Block Ciphers. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 31–46. Springer, 2002.
10. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. Series on Discrete Mathematics and its Applications. CRC Press, 1997. ISBN 0-8493-8523-7, Available online at `http://www.cacr.math.uwaterloo.ca/hac/`.
11. National Institute of Standards and Technology. AES Competition Website (archived). `http://csrc.nist.gov/archive/aes/index.html`.
12. National Institute of Standards and Technology. SHA-3 Competition Website. `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.